Técnicas Avanzadas de Programación

Practica Clase 2

Diseño y Análisis de Algoritmos - Notación Big O

Alumnos:
Oyarzo, Mariana Belen
Pastor Neil, Pablo Daniel Alberto

a) Encontrar el número más frecuente en una lista.
b) Verificar si una lista de enteros es palíndroma.
c) Contar la cantidad de elementos únicos en una lista.
d) Determinar si dos listas contienen los mismos elementos (sin importar el orden ni las repeticiones).

Encontrar el número más frecuente en una lista:

Pseudocodigo:

```
Algoritmo masfrecuente_Tecnicas
    Definir mi_cad, digito, masrepetido como cadena
    Definir contador, maxcontador, i, j como entero
    contador←0
    masrepetido←""
    Escribir "Ingrese su lista de numeros:"
    leer mi_cad
    //para ver cual se repite mas
        Para i←1 hasta Longitud(mi_cad) Hacer//recorremos
            digito←subcadena(mi_cad,i,i)
            Contador←0
            //contar cuantas veces aparece el dig
            Para j←0 hasta longitud(mi_cad) Hacer//cuantas veces
                si subcadena(mi_cad,j,j)= digito entonces
                    Contador←Contador+1
                FinSi
            FinPara
            //guarad el que mas veces aparece
            Si contador > maxcontador Entonces
                maxcontador←contador
                masrepetido←digito
            FinSi
        FinPara
    Escribir "El número que más se repite es: ", masRepetido, ",  ", maxcontador, " veces."
FinAlgoritmo

```

Código en python:

```
def ejercicio1(listaAEvaluar): #Nro mas frecuente
    #Creamos una lista a mano para evaluar y definimos otra como contador,
    #del tamaño del elemento de valor máximo de la lista a evaluar
    listaContador=[0] * (int((max(listaAEvaluar))) + 1)                    # 1

    #Recorremos la lista a evaluar e incrementamos en 1 el elemento de la posicion
    # de la lista contador que corresponde con el valor del elemento de la lista a evaluar
    for x in listaAEvaluar:                                                # n
        listaContador[x]= int(listaContador[x]) +1                         # n

    #Recorremos la lista contador para obtener la posicion del elemento de valor mayor
    #Dicha posicion se corresponde con el maximo valor de los elementos de la que evaluamos
    maximo=-1                                                              # 1
    for x in range(len(listaContador)):                                    # n
        if x>maximo: maximo = x                                            # n

    #Informamos y para la posteridad retornamos el valor.
    print('El valor mas grande de la lista es: ' + str(x))                 # 1
    return x                                                               # 1
                                                                #----------------
                                                                    # 4n+5 => n
```

Salidas y tiempos con distintos parametros:

```
 94
 95  def main():
 96      #Para verificar el tiempo de ejecucion mostramos
 97      print(datetime.datetime.now())
 98      ejercicio1([4,5,9,9,9,5])
 99  #    ejercicio1([4,3])
100      # ejercicio2([4, 7, 3, 3, 7, 4])
101      # ejercicio2([4, 7, 3, 3, 9, 4])
102      # ejercicio3([4, 9, 5, 3,4 , 7])
103      # ejercicio3([4, 7,7,7,7,7,7,7,7,7,7,7])
104      # ejercicio4([4, 9, 5, 3, 4 ,7], [4, 9, 6, 3, 4 ,
105      # ejercicio4([4, 7, 5, 3, 4 ,7], [4, 7, 5, 3, 9 ,
106      # ejercicio4([4, 7, 5], [4, 7, 5])
107      print(datetime.datetime.now())
108
109  main()
110
111
112
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
lo/Desktop/Uni/1º 2\2 Tecnicas avanzadas de programacion/TP
s - Notación Big O.py"
2025-08-16 21:09:52.660139
El valor mas grande de la lista es: 9
2025-08-16 21:09:52.660262
pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
```

```
 94
 95  def main():
 96      #Para verificar el tiempo de ejecucion mostramos
 97      print(datetime.datetime.now())
 98      # ejercicio1([4,5,9,9,9,5])
 99      ejercicio1([4,3])
100      # ejercicio2([4, 7, 3, 3, 7, 4])
101      # ejercicio2([4, 7, 3, 3, 9, 4])
102      # ejercicio3([4, 9, 5, 3,4 , 7])
103      # ejercicio3([4, 7,7,7,7,7,7,7,7,7,7,7])
104      # ejercicio4([4, 9, 5, 3, 4 ,7], [4, 9, 6, 3, 4 ,
105      # ejercicio4([4, 7, 5, 3, 4 ,7], [4, 7, 5, 3, 9 ,
106      # ejercicio4([4, 7, 5], [4, 7, 5])
107      print(datetime.datetime.now())
108
109  main()
110
111
112
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
lo/Desktop/Uni/1º 2\2 Tecnicas avanzadas de programacion/TP
s - Notación Big O.py"
2025-08-16 21:10:37.212402
El valor mas grande de la lista es: 4
2025-08-16 21:10:37.212495
pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
```

Verificar si una lista de enteros es palíndroma.

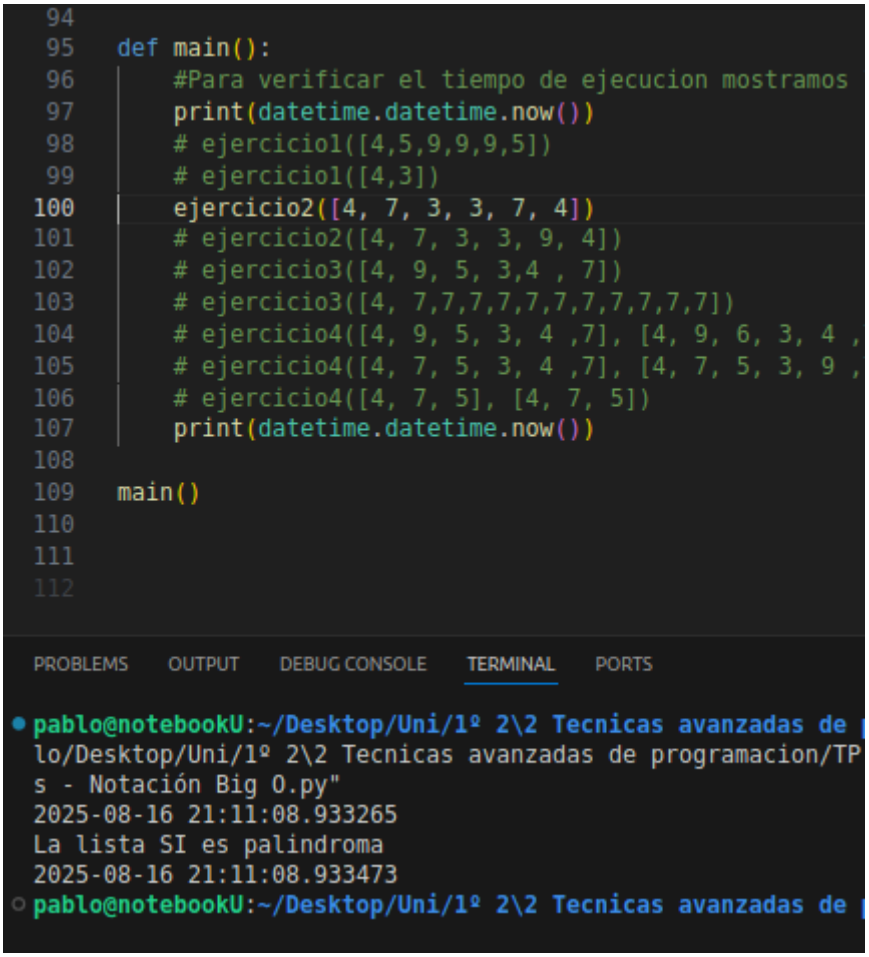
Pseudocodigo:

```
Algoritmo Iec_avanzadasPalindroma
    Definir mi_cad, alreves como cadena
    Escribir"Ingrese lista numeros "
    Leer mi_cad

    alreves← "" //Para guardar la caden al reves
    para i←longitud(mi_cad) hasta 0 con paso -1 hacer //desde el final a 0, el princ
        alreves←alreves+subcadena(mi_cad, i, i) //guardamos al reves
    FinPara

    Si mi_cad=alreves Entonces
        Escribir "  La lista es palindroma"
    sino
        Escribir "La lista no es palindroma"
    Finsi
FinAlgoritmo
```

Algoritmo en Python:

```
def ejercicio2(listaAEvaluar):
    #De una lista definida en el script, con un solo ciclo de iteracion comparamos
    #sus elementos desde los extremos hasta el centro de la misma.
    for x in range(len(listaAEvaluar)):                                    # n

        #En caso de discrepancia se informa que no es palindroma y se retorna falso.
        if listaAEvaluar[x] != listaAEvaluar[len(listaAEvaluar) - x-1]:    # n
            print('La lista NO es palindroma')                             # n
            return False                                                   # n
        #al alcanzarse el centro de la lista se rompe el loop
        if x >= len(listaAEvaluar)-x-1: break                              # n
    #al finalizar el bucle se informa y se retorna verdadero
    print('La lista SI es palindroma')                                     # 1
    return True                                                            # 1
                                                                  #----------------
                                                                     # 5n +3 -> n
```

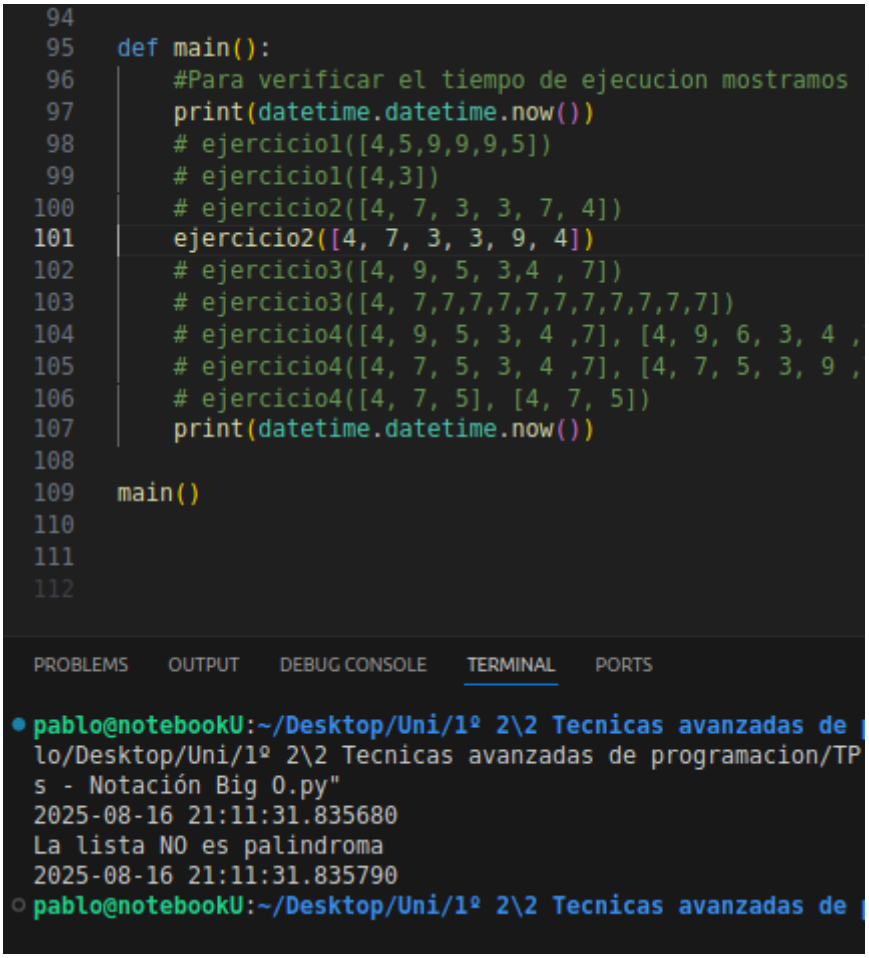
Salida con distintos argumentos y tiempos de ejecución correspondiente:



```
 94
 95  def main():
 96      #Para verificar el tiempo de ejecucion mostramos
 97      print(datetime.datetime.now())
 98      # ejercicio1([4,5,9,9,9,5])
 99      # ejercicio1([4,3])
100      ejercicio2([4, 7, 3, 3, 7, 4])
101      # ejercicio2([4, 7, 3, 3, 9, 4])
102      # ejercicio3([4, 9, 5, 3,4 , 7])
103      # ejercicio3([4, 7,7,7,7,7,7,7,7,7,7,7])
104      # ejercicio4([4, 9, 5, 3, 4 ,7], [4, 9, 6, 3, 4 ,
105      # ejercicio4([4, 7, 5, 3, 4 ,7], [4, 7, 5, 3, 9 ,
106      # ejercicio4([4, 7, 5], [4, 7, 5])
107      print(datetime.datetime.now())
108
109  main()
110
111
112
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
lo/Desktop/Uni/1º 2\2 Tecnicas avanzadas de programacion/TP
s - Notación Big O.py"
2025-08-16 21:11:08.933265
La lista SI es palindroma
2025-08-16 21:11:08.933473
pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
```



```
 94
 95  def main():
 96      #Para verificar el tiempo de ejecucion mostramos
 97      print(datetime.datetime.now())
 98      # ejercicio1([4,5,9,9,9,5])
 99      # ejercicio1([4,3])
100      # ejercicio2([4, 7, 3, 3, 7, 4])
101      ejercicio2([4, 7, 3, 3, 9, 4])
102      # ejercicio3([4, 9, 5, 3,4 , 7])
103      # ejercicio3([4, 7,7,7,7,7,7,7,7,7,7,7])
104      # ejercicio4([4, 9, 5, 3, 4 ,7], [4, 9, 6, 3, 4 ,
105      # ejercicio4([4, 7, 5, 3, 4 ,7], [4, 7, 5, 3, 9 ,
106      # ejercicio4([4, 7, 5], [4, 7, 5])
107      print(datetime.datetime.now())
108
109  main()
110
111
112
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
lo/Desktop/Uni/1º 2\2 Tecnicas avanzadas de programacion/TP
s - Notación Big O.py"
2025-08-16 21:11:31.835680
La lista NO es palindroma
2025-08-16 21:11:31.835790
pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
```

Contar la cantidad de elementos únicos en una lista.

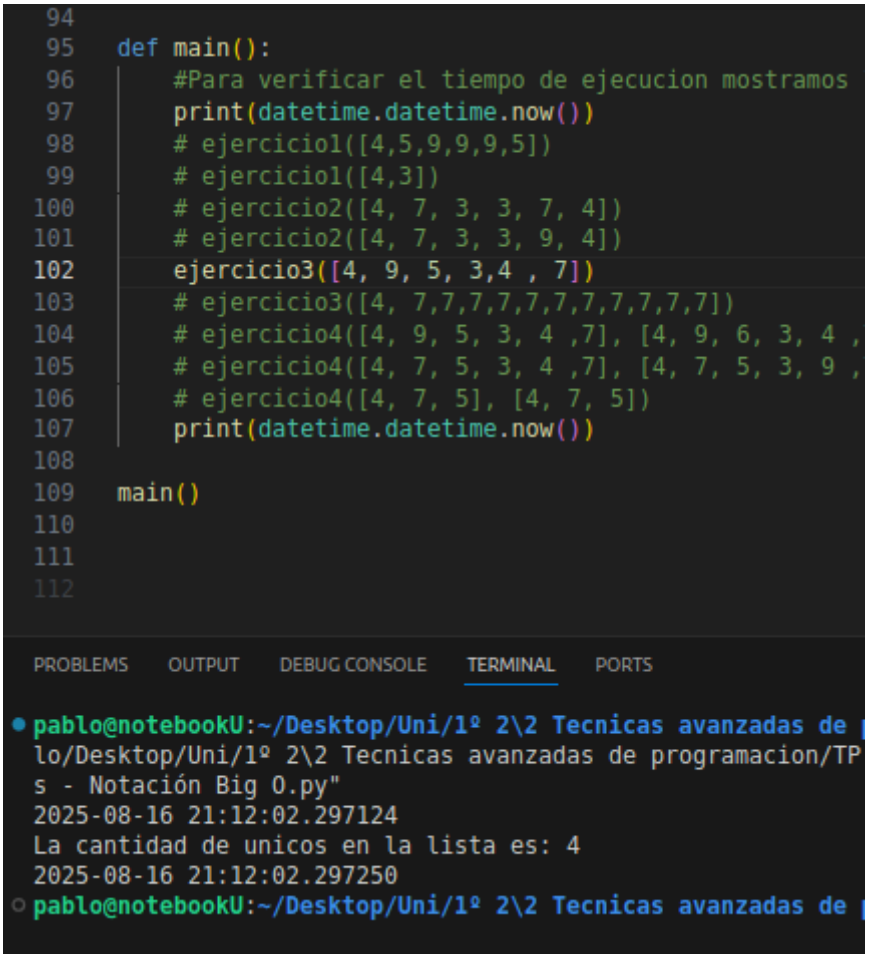
Pseudocodigo:

```
Algoritmo elemUnicos_tecnicas
    Definir mi_cad, vistos, digito como cadena
    definir i, unicos como entero
    Definir existe como logico
    unicos←0
    vistos←""

    Escribir"Ingrese lista de numeros: "
    leer mi_cad
    //verif si ya esta en vistos

Para i←0 Hasta longitud(mi_cad) Hacer
    digito← Subcadena(mi_cad,i,i)
    existe←falso
    Para j ←0 hasta longitud(vistos) Hacer
        si subcadena(vistos,j,j)= digito entonces
            existe=verdadero
        FinSi
    FinPara


si existe=Falso entonces
    unicos←unicos+1
    vistos←vistos+digito
FinSi
FinPara

Escribir "La cantidad de elementos unicos en la lista es: ", unicos " y son los digitos: ", vistos
FinAlgoritmo
```

Codigo en piton:

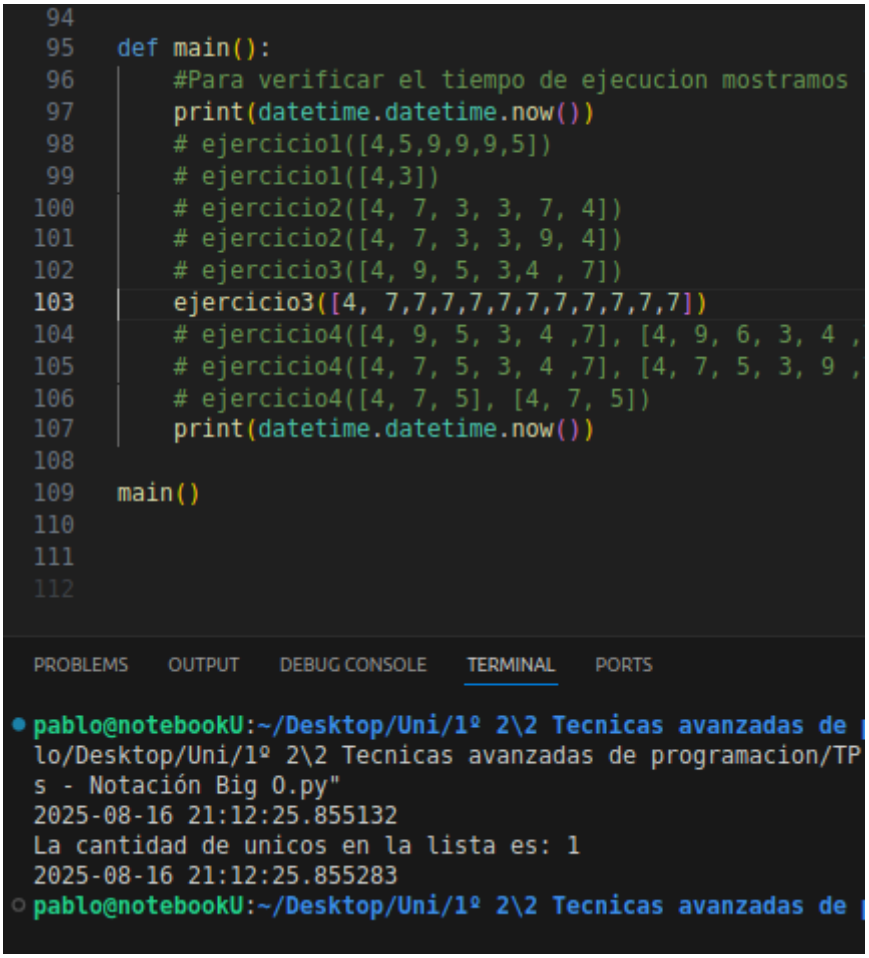
```
def ejercicio3(listaAEvaluar):
    #Para contar unicos se realizan 2 ciclos de iteracion, para comparar c/ elemento
    #contra el resto. Se marca en verdadero un flag, el cual se niega al encontrarse una
    #igualdad de elementos. Si al terminar el ciclo interior el flag de unico
    # se encuentra en ferdadero se aumenta en 1 el contador de unicos. en caso de ser
    # falso, no se incrementa. Al terminar se notifica y retorna la cantidad
    cantidadDeUnicos=0                                                          # 1
    for a in range(len(listaAEvaluar)):                                         # n
        unico = True                                                            # n
        for b in range(len(listaAEvaluar)):                                     # n²
            if listaAEvaluar[a] == listaAEvaluar[b] and a != b: unico = False   # n²
        if unico: cantidadDeUnicos+=1                                           # n
    print('La cantidad de unicos en la lista es: ' + str(cantidadDeUnicos))    # 1
    return cantidadDeUnicos                                                     # 1
                                                                            #-------------
                                                                            # 2n² + 3n + 4  -> n²
```

Salid con diferenter parametros y sus tiempos de ejecucion:



```
 94
 95  def main():
 96      #Para verificar el tiempo de ejecucion mostramos
 97      print(datetime.datetime.now())
 98      # ejercicio1([4,5,9,9,9,5])
 99      # ejercicio1([4,3])
100      # ejercicio2([4, 7, 3, 3, 7, 4])
101      # ejercicio2([4, 7, 3, 3, 9, 4])
102      ejercicio3([4, 9, 5, 3,4 , 7])
103      # ejercicio3([4, 7,7,7,7,7,7,7,7,7,7,7])
104      # ejercicio4([4, 9, 5, 3, 4 ,7], [4, 9, 6, 3, 4 ,
105      # ejercicio4([4, 7, 5, 3, 4 ,7], [4, 7, 5, 3, 9 ,
106      # ejercicio4([4, 7, 5], [4, 7, 5])
107      print(datetime.datetime.now())
108
109  main()
110
111
112
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
lo/Desktop/Uni/1º 2\2 Tecnicas avanzadas de programacion/TP
s - Notación Big O.py"
2025-08-16 21:12:02.297124
La cantidad de unicos en la lista es: 4
2025-08-16 21:12:02.297250
pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
```



```
 94
 95  def main():
 96      #Para verificar el tiempo de ejecucion mostramos
 97      print(datetime.datetime.now())
 98      # ejercicio1([4,5,9,9,9,5])
 99      # ejercicio1([4,3])
100      # ejercicio2([4, 7, 3, 3, 7, 4])
101      # ejercicio2([4, 7, 3, 3, 9, 4])
102      # ejercicio3([4, 9, 5, 3,4 , 7])
103      ejercicio3([4, 7,7,7,7,7,7,7,7,7,7,7])
104      # ejercicio4([4, 9, 5, 3, 4 ,7], [4, 9, 6, 3, 4 ,
105      # ejercicio4([4, 7, 5, 3, 4 ,7], [4, 7, 5, 3, 9 ,
106      # ejercicio4([4, 7, 5], [4, 7, 5])
107      print(datetime.datetime.now())
108
109  main()
110
111
112
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
lo/Desktop/Uni/1º 2\2 Tecnicas avanzadas de programacion/TP
s - Notación Big O.py"
2025-08-16 21:12:25.855132
La cantidad de unicos en la lista es: 1
2025-08-16 21:12:25.855283
pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
```

Determinar si dos listas contienen los mismos elementos (sin importar el orden ni las repeticiones).

Pseudocodigo:

```
Algoritmo CompararCadenas_Tecnicas
    Definir cad1, cad2 Como Cadena
    Definir i Como Entero
    Definir letra Como Caracter
    Definir mismo Como Logico
    Escribir "Ingrese la primera cadena: "
    Leer cad1
    Escribir "Ingrese la segunda cadena: "
    Leer cad2

    mismo ← Verdadero

    // Recorremos cada caracter de cad1
    Para i ← 0 Hasta Longitud(cad1) Hacer
        letra ← Subcadena(cad1,i,i)

        // Verificamos que la cantidad de veces que aparece
        // en cad1 sea igual que en cad2
        Si ContarCaracter(cad1,letra) ≠ ContarCaracter(cad2,letra) Entonces
            mismo ←Falso
        FinSi
    FinPara

    // También recorremos cad2 para no perder caracteres
    Para i ← 0 Hasta Longitud(cad2) Hacer
        letra ← Subcadena(cad2,i,i)

        Si ContarCaracter(cad1,letra) ≠ ContarCaracter(cad2,letra) Entonces
            mismo ← Falso
        FinSi
    FinPara

    Si mismo Entonces
        Escribir "Las cadenas tienen el mismo contenido."
    SiNo
        Escribir "Las cadenas no tienen el mismo contenido."
    FinSi
FinAlgoritmo


Funcion resultado ← ContarCaracter(cadena, letra)
    Definir i, resultado Como Entero
    resultado ← 0
    Para i ← 0 Hasta Longitud(cadena) Hacer
        Si Subcadena(cadena,i,i) = letra Entonces
            resultado ← resultado + 1
        FinSi
    FinPara
FinFuncion
```

en Python:

```python
def ejercicio4(listaA, listaB):
    #Se definen 2 listas a comprar. al anidar 2 bucles vamos comparando c/elemento
    # de una lista contra todos los de la otra. en caso de que exista coincidencia
    # se pone en true un flag y se rompe el ciclo de comparacion contra ese elemento
    # en caso de que no exista ningun elemento igual en la 2da lista se informa y
    # se retorna falso. Luego se repite el proceso con los parametros invertidos.
    # en caso de que todos los elementos existan entre listas se
    # avisa y alcanza el return true del final.


    for a in listaA:                                                    # n
        existe = False                                                  # n
        for b in listaB:                                                # n²
            if a == b :                                                 # n²
                existe = True                                           # n²
                break                                                   # n²
        if not existe :                                                 # n
            print('El elemento ' + str(a) + ' del 1er arg no existe en el 2do. Fin')  # n
            return False                                                # n
    for b in listaB:                                                    # n
        existe = False                                                  # n
        for a in listaA:                                                # n²
            if a == b :                                                 # n²
                existe = True                                           # n²
                break                                                   # n²
        if not existe :                                                 # n
            print('El elemento ' + str(b) + ' del 2do arg no existe en el 1ro. Fin')  # n
            return False                                                # n
    print('Ambas listas poseen los mismos elementos')                   # 1
    return True                                                         # 1
                                                        #------------------
                                                        # 8n² + 9n + 2 -> n²
```

Salida con distintos parametros y sus tiempos:

```python
def main():
    #Para verificar el tiempo de ejecucion mostramos
    print(datetime.datetime.now())
    # ejercicio1([4,5,9,9,9,5])
    # ejercicio1([4,3])
    # ejercicio2([4, 7, 3, 3, 7, 4])
    # ejercicio2([4, 7, 3, 3, 9, 4])
    # ejercicio3([4, 9, 5, 3,4 , 7])
    # ejercicio3([4, 7,7,7,7,7,7,7,7,7,7,7])
    ejercicio4([4, 9, 5, 3, 4 ,7], [4, 9, 6, 3, 4 ,7]
    # ejercicio4([4, 7, 5, 3, 4 ,7], [4, 7, 5, 3, 9 ,
    # ejercicio4([4, 7, 5], [4, 7, 5])
    print(datetime.datetime.now())

main()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
lo/Desktop/Uni/1º 2\2 Tecnicas avanzadas de programacion/TP
s - Notación Big O.py"
2025-08-16 21:13:19.975199
El elemento 5 del 1er arg no existe en el 2do. Fin
2025-08-16 21:13:19.975284
pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
```

```python

def main():
    #Para verificar el tiempo de ejecucion mostramos
    print(datetime.datetime.now())
    # ejercicio1([4,5,9,9,9,5])
    # ejercicio1([4,3])
    # ejercicio2([4, 7, 3, 3, 7, 4])
    # ejercicio2([4, 7, 3, 3, 9, 4])
    # ejercicio3([4, 9, 5, 3,4 , 7])
    # ejercicio3([4, 7,7,7,7,7,7,7,7,7,7,7])
    # ejercicio4([4, 9, 5, 3, 4 ,7], [4, 9, 6, 3, 4 ,
    ejercicio4([4, 7, 5, 3, 4 ,7], [4, 7, 5, 3, 9 ,7]
    # ejercicio4([4, 7, 5], [4, 7, 5])
    print(datetime.datetime.now())

main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
lo/Desktop/Uni/1º 2\2 Tecnicas avanzadas de programacion/TP
s - Notación Big O.py"
2025-08-16 21:14:59.180258
El elemento 9 del 2do arg no existe en el 1ro. Fin
2025-08-16 21:14:59.180384
pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
```

```python

def main():
    #Para verificar el tiempo de ejecucion mostramos
    print(datetime.datetime.now())
    # ejercicio1([4,5,9,9,9,5])
    # ejercicio1([4,3])
    # ejercicio2([4, 7, 3, 3, 7, 4])
    # ejercicio2([4, 7, 3, 3, 9, 4])
    # ejercicio3([4, 9, 5, 3,4 , 7])
    # ejercicio3([4, 7,7,7,7,7,7,7,7,7,7,7])
    # ejercicio4([4, 9, 5, 3, 4 ,7], [4, 9, 6, 3, 4 ,
    # ejercicio4([4, 7, 5, 3, 4 ,7], [4, 7, 5, 3, 9 ,
    ejercicio4([4, 7, 5], [4, 7, 5])
    print(datetime.datetime.now())

main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
lo/Desktop/Uni/1º 2\2 Tecnicas avanzadas de programacion/TP
s - Notación Big O.py"
2025-08-16 21:15:15.985124
Ambas listas poseen los mismos elementos
2025-08-16 21:15:15.985231
pablo@notebookU:~/Desktop/Uni/1º 2\2 Tecnicas avanzadas de
```

Tabla comparativa funcion + parametros / diff de tiempos de ejec.:
(el analisis de la complejidad temporal con Big O se encuentra en las capturas)

Ejercicio / Parametro	t inicio [µs]	t fin [µs]	Diff [µs]
ejercicio1([4,5,9,9,9,5])	139	262	123
ejercicio1([4,3])	402	495	93
ejercicio2([4, 7, 3, 3, 7, 4])	265	473	208
ejercicio2([4, 7, 3, 3, 9, 4])	680	790	110
ejercicio3([4, 9, 5, 3,4 , 7])	124	250	126
ejercicio3([4, 7,7,7,7,7,7,7,7,7,7,7])	132	283	151
ejercicio4([4, 9, 5, 3, 4 ,7], [4, 9, 6, 3, 4 ,7])	199	284	85
ejercicio4([4, 7, 5, 3, 4 ,7], [4, 7, 5, 3, 9 ,7])	258	384	126
ejercicio4([4, 7, 5], [4, 7, 5])	124	231	107

¿Cuál de los algoritmos fue más eficiente y por qué?
¿Qué dificultades surgieron al estimar la complejidad Big O?
¿Cómo ayuda este tipo de análisis en la toma de decisiones de programación?

Los algoritmos 1 y 2 son mas eficientes que los ultimos, ya que, en los primeros el tiempo de ejecucion es lineal(O(n)) con respecto a la entrada, sin embargo en los finales es cuadratico ($O(n^2)$) con respecto a su input.

La complejidad que tuvimos fue al hilar fino sobre las condiciones en bucles con condicionales, cuando estos en su peor caso tienden a O(n) en vez de $O(n^2)$.

Este tipo de análisis permite estimar lo considerable del tiempo de ejecucion cuando la entrada de los programas posee un gran volumen de datos, detectando de forma temprana la necesidad de optimizacion.